Design Instrumental Web Services for Online Experiment Systems

Yan, Y.; Liang, Y.; Du, X.; Saliah-Hassane, H.; Ghorbani, A.

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de technologie de l'information

NRC · CNRC

Design Instrumental Web Services for Online Experiment Systems *

Yan, Y., Liang, Y., Du, X., Saliah-Hassane, H., Ghorbani, A.
June 2005

* published at Ed-MEDIA2005: World Conference on Education Hypermedia and Telecommunication. Montreal, Québec, Canada. June 27-July 2, 2005. NRC 48221.

Canada

Design Instrumental Web Services for Online Experiment Systems

Yuhong Yan
NRC-IIT, Fredericton, NB, Canada
Yuhong.yan@nrc.gc.ca

Xinge Du
Faculty of Computer Science, UNB, Canada
Xinge.Du@unb.ca

Hamadou Saliah-Hassane
Télé-université, Montreal, Canada
saliah@teluq.uquebec.ca

Yong Liang
Faculty of Computer Science, UNB, Canada
Yong.liang@unb.ca

Ali Ghorbani
Faculty of Computer Science, UNB, Canada
ghorbani@unb.ca

Abstract: Web services add a language-independent layer for distributed applications. It is well motivated to use Web Services for e-science and e-learning applications, where resources-sharing is a demand. Online experimentation is one of the e-learning applications. Web Services have intrinsic weaknesses on latency and scalability due to more transport layers used. This paper presents our experiences in using Web services for prototyping a service-oriented architecture for Online Experiment Systems (OES). We present the methodology to wrap the functions of experimental devices into web services and the advance features the instrument web services should have. We benchmark the performance of this system when using SOAP as a wire format for Web Services. Our results show the degradation of the performance due to a large amount data to transfer. We present the solutions to optimize the performance.

1. Introduction

Web services are the latest technology for distributed applications. The services are provided by software components over the Internet. The services are invoked by sending eXtensible Markup Language (XML) based Simple Object Access Protocol message (SOAP) to the remote components. Web services rely on internet protocols, such as Hypertext Transfer Protocol (HTTP), Blocks Extensible Exchange Protocol (BEEP) and XML technology to ensure the *interoperability* of the components on different platforms and/or implemented in different programming languages. W3C accepts the following standards: SOAP is a message-based communication for component interaction (W3C, 2004a); Web Service Description Language (WSDL), component interface definition (W3C, 2004b), and; Universal Description, Discovery Integration (UDDI.org, 2004), service discovery and integration (UDDI.org, 2004).

In the scope of e-learning, the online experiment system is to provide services for the students to conduct experiments via the Internet. Typically the students can operate remote instruments via GUI of the online experiment system. Online experiment system is a typical distributed application. Web services are replacing the classic client/server architecture to be the architecture of online experiment systems. The heterogeneous resources, such as instruments, testing devices, or one entire experiment, which are scattered over the Internet, can be wrapped as individual web services. Through interfaces in UDDI, these services can be discovered and integrated. At the runtime, the WSDL determines how the arguments are transferred to invoke the services. The online experiment system is able to conduct the operations to discover and integrate these services automatically. This is seamless to

the users so that they would not notice that the instruments and devices are from different physical locations. The Internet is a bus connecting all these services. Service providers can charge for services provided based on user requests.

Although Web Services have strong advantages on interoperability, Web Services have intrinsic weaknesses on latency and scalability due to more transport layers used. For online experimentation, this would cause problems. In this paper, we present our experiences in using web services for prototyping a service-oriented architecture for Online Experiment Systems (OES). We present the methodology to wrap the functions of experimental devices into web services. We propose some advance features the instrument web services should have. We then benchmark the performance of this kind of system which using SOAP as a wire format for Web Services. Our results show the degradation of the performance due to a large amount data to transfer using SOAP. We present a set of solutions to optimize the performance.

This paper is organized as following: section 2 presents the general framework; section 3 designs the web services for instrument services; section 4 benchmarks web services for this application; section 5 presents the optimization methods to improve the performance; and section 6 is the conclusion.

2. The Web Service-based Framework for Online Experiment System

An Online Experiment System (OES) uses the scattered computational resources and instrument services on the networks for experiments. The online laboratory system we present here is a *web enabled distributed system*. It has two fold meanings: the user accesses the online experiment system via the web interface; the heterogeneous resources and devices interoperate with each other via Web services standards. The goals of this framework are: 1) sharing the experimental resources among different labs via the Internet; 2) increasing the ability of computation and sharing data among different labs, and; 3) enabling users to access online labs any time and from anywhere either for self-learning or collaborative laboratory work sessions.

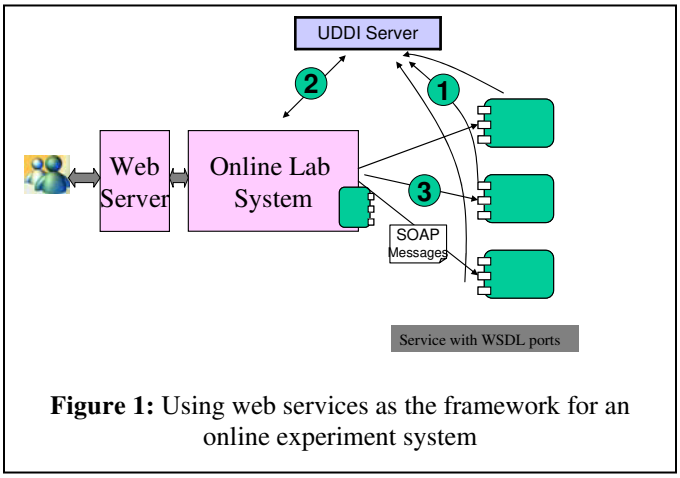

Figure 1: Using web services as the framework for an online experiment system

The above figure (Fig. 1) shows how the system using web services works. Its front-end is web-based, i.e. a web server is used to render the GUI interface (see the next section). Its back-end has the functions to manage the users' information and the operations of the experiments. Most importantly, the backend can use scattered resources on the Internet for one experiment. For example, it can use instruments and devices from different online laboratories for one experiment, and it can use heterogeneous computational resources to process the data generated from the experiment. The user uses web browser to connect to the OES. The OES uses remote services based on Web Services technology. The online experiment and the remote services use WSDL to describe the operational interfaces.

Web Services serve as the transport layer of the system. SOAP messages are sent to invoke a service. These protocols are widely accepted by different operating systems and implemented by different programming languages. Therefore, the interoperability is ensured. To make the system work, a service provider first registers its services in a registry server (step 1 in Fig. 1). Web services use UDDI server, while Grid Services use Index Server. We expect the two standards will be merged. Otherwise, our system will accept both standards. A service requester searches the

registry server and gets all the potential resources. It selects the proper services based on its own criteria (step 2). The service requestor sends SOAP messages directly to the service provider to invoke the remote service (step 3).

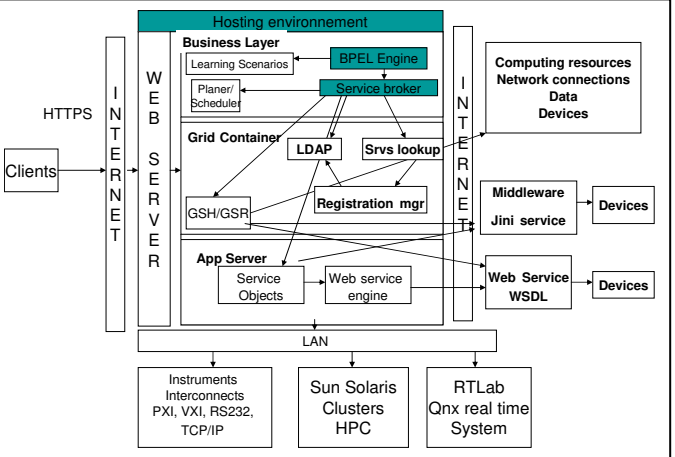

Figure 2: Architecture of the online experiment system

In Figure 2, the block below the "hosting environment" is the online laboratory system. It uses a web server for the front-end representation. The back-end has three layers. The top layer is the logic layer, where the learning scenarios are defined and the processes are managed. The learning scenarios are defined in four aspects (Paguette, 1999): learning objects, a pedagogical model, a media model and distribution. Among those, the pedagogical model defines the process of a course. The process is translated directly into Business Process Execution Language (BPEL) (Andrews, *et al.*, 2004). The ***BEPL engine*** is a tool to monitor and control the process automatically. The BEPL engine is able to automatically invoke remote web services. The activities in a learning scenario may need remote web services. The ***Service Broker*** determines if the services come from local services (e.g. the blocks under the "*LAN*"), or remote external services (e.g. the blocks of "*jini services*", "*web services*"). Service Broker knows the different protocols the remote services use. For Grid services, it sends the requests to the GSH/GSR (Grid Service Handler/Grid Service References) in the ***Grid Container***. GSH/GSR is a mechanism in Grid Service to get the reference of the remote objects and forward the requests to the remote objects. GSH/GSR is able to invoke the services either in middleware, (e.g. jini), or in web services. Service Broker can also invoke web services without the GSH/GSR interface by sending the request to the service objects in the application server (the bottom layer). Service Broker regularly calls the ***Service Lookup*** ("*srv lookup*" in Figure 2) and updates the local ***LDAP*** with the results. ***Registration Manager*** ("*Registration Mrg*" in Figure 2) helps to convert information from a service registry into LDAP. The bottom layer is the ***Application Server*** layer. The Application Server provides flexible mechanisms to manage the ***Service Objects*** and interface to the ***Web Service Engine***. Service Objects are some software components that process the data from remote web services. See the next section for one example of service implementation. The Web Service Engine sends the SOAP message to invoke the remote web services. This framework works with the computing resources using Grid protocols, software components using middleware, and web services components. Thus we think it covers all the resources needed for online experiments. As Grid Services will merge with Web Services in the future, we believe that the two lower layers in Figure 2 will, at some point, be united into one layer.

3. Design an instrument web service

An instrument service is a basic service for OES. Most of the efforts in this paper are to present the methodology to wrap the instrument service into a web service, and to discuss the performance of using Web Services for this application.

3. 1 Design the WSDL Interface for the Instrument Service

The WSDL file describes the operations on the instrument and the arguments to invoke the operations. The commands to the instrument are encoded as SOAP messages. The results from the instruments are encoded as well as SOAP messages. The WSDL file can be generated from implementation code using some automatic tools. Since we do not limit our methodology to a specific programming language, we do not talk about APIs. Instead, we talk about the interfaces defined in WSDL.
In order to invoke an instrument service, we need to know at least four kinds of information: 1) the input parameters to operate the instrument; 2) the output results from the instrument; 3) the information about the availability of the instrument; and 4) the xml file to describe the instrument panel. We also define two optional interfaces: 5) the metadata complaint to LOM standard; and 6) the quality of service (QoS). Though the operational interfaces are not necessary information to operate the instrument, they provide important information for service discovery and integration.

3.1.1 *Input/output Interfaces*

Input/output interfaces are determined by the API to operate the instrument. There is automatic tool to map the input/output arguments into WSDL descriptions. Here we demonstrate how it works using a type of oscilloscope from National Instrument as an example.

An oscilloscope is used to monitor a waveform from an arbitrary waveform generator. The set of control parameters are "*amplitude*", "*cycles*", "*phase*" and "*sampling rate*". The output is a double array which represents the waveform. The snippet of the WSDL to describe the input/output interfaces is in Table 1. The operation ***getWaveform*** is an *in-out* type service, which means it receives the input arguments embedded in SOAP message and immediately returns the results in response SOAP message. The request SOAP message is defined in the message named ***getWaveformRequest***. It takes in three *double* typed parameters which are correspondent to "*amplitude*", "*cycles*", and "*phase*" respectively. The fourth parameter is *int* typed which is correspondent to "*sampling rate*". The response SOAP message is defined in the message named ***getWaveformResponse***. It defines a two dimension array of *doubles*.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions .....>
      ......
  <!--define the response message -->
  <wsdl:message name="getWaveformResponse">
    <wsdl:part name="getWaveformReturn" type="impl:ArrayOf_tns2_double"/>
  </wsdl:message>
 <!--define the request message -->
  <wsdl:message name="getWaveformRequest">
    <wsdl:part name="amp" type="xsd:double"/>
    <wsdl:part name="numc" type="xsd:double"/>
    <wsdl:part name="phase" type="xsd:double"/>
    <wsdl:part name="sampr" type="xsd:int"/>
  </wsdl:message>
  <!--define the operation -->
 <wsdl:operation name="getWaveform">
  <wsdl:input name="getWaveformRequest">
  </wsdl:input>
  <wsdl:output name="getWaveformResponse">
  </wsdl:output>
</wsdl:operation>
      ......
</wsdl:definitions>
```

Table 1: the snippet of WSDL to operate an oscilloscope

3.1.2 *Interfaces of Other Information*

The other information about an instrument, such as the availability, LOM information, and QoS can be defined in XML files. In the WSDL, we define the operations to download the xml files. These operations are *out* typed operations (i.e. only have response SOAP message). Table 2 displays these operations in WSDL.

Next, we show what information is defined in the xml file. IEEE LOM standard is to describe a learning object (IEEE, 1999). But when using it to describe an instrument, the vocabulary is not the same. Some information especially for instruments, such as availability, is not in LOM standard. So we use the extended LOM standard for experimentation context developed in (Bagnasco, Chirico, and Scapolla, 2002). The availability is important for booking the instrument. We have a separated operation just to get this information. The user can list all the available timeslot during a time interval for a week or so, or query if the instrument is available for a specific minimal time unit.

```
<!--define the operation -->
<wsdl:operation name="getMetaData">
  <wsdl:output name="getMetaDataResponse">
  </wsdl:output>
<!--define the operation -->
<wsdl:operation name="getAvailabilityInfo">
  <wsdl:output name="getAvailabilityResponse">
  </wsdl:output>
<!--define the operation -->
<wsdl:operation name="getQoSInfo">
  <wsdl:output name="getQoSResponse">
  </wsdl:output>

</wsdl:operation>
```

Table 2: the operations to get metadata information in WSDL

QoS information is accumulated from history, and can become an important selling and differentiating point of web services with similar functionality. We record the successful rate to connect to the instrument, the response time to the instrument, and customers rating to use its service. QoS information is used when selecting available instruments for an experiment. The higher QoS of the instrument service, the more likely the OES selects this instrument or recommends it to the user to use.

The information about the panel of the instrument is also defined in xml. Because it involves interpret and renders the information in GUI, we need more work on it. The next section will deal with the GUI only.

3.2 Design the Web GUI for the Instrument

An instrument has its individualized panel. In the Learning Object Repository Network (LORNET) project, we studied how to display the panel as a java distributed application (Fattouh and Saliah, 2004). DMM is a XML schema to define the syntax of an instrument panel. An XML file compliant to the DMM is a description of the panel. In (Fattouh and Saliah, 2004), the XML file is parsed by JAXB, and its components are mapped to java AWT components. The system is implemented using Jini technology. The XML is downloaded from the Jini registration server and displayed on client side as a Java application. The user can operate the remote object from the GUI interface. In (Saliah, *et al.*, 2005), we present a similar method to (Fattouh and Saliah, 2004). Instead of using AWT classes, we map the panel objects to JSP objects at the server side. The client just needs to require a normal web page, instead of using any jini code. Therefore, we have a thin client. If we need to show arbitrary shapes, such as waveform, it is a little bit more complex. We can have two options. If we want to achieve zero installation at the client side, i.e. no code to be installed, we can generate the jpg image for the waveform. It is a matured technology. Or if we allow the client side to use applets (for java) or activeX control (for windows platform), we have a thicker client. It needs very simple coding.

3.3 The Advanced Requirements for Instrument Services

Unlike most existing web services whose main propose is to provide information, the instrument web services involve remotely operating real devices in real time. Improper design of the web services can cause the damage of the instrument, false measurement and control, which fails the online experiment. Therefore we present the special requirements for the instrument web services. By using proper software technologies, these requirements can be satisfied.

3.3.1 *Stateful Service*

A stateful service keeps client specific state between method invocations. There are two scenarios which need a stateful service: 1) when it needs to record the operations from one user and control how the user can use the service; 2) when there is a series of actions that data needs to be recorded and transferred between the actions. Since we prefer to use open source software, our choices are between java serlvet and Enterprise Java Bean (EJB). EJB is more scalable and has more inherited functions, such as transactions and persistence, than java servlet, while servlet is lighter and faster. So except some necessary situation, our first choice is serlvet.

3.3.2 *Server Side Reliability Mechanisms*

The latency of the network can cause the difficulties in real-time control and measurement. The lost of the network connections can cause the lost of control of the remote instruments. Sometimes it can even cause physical damages. In the context of e-learning, we do not deal with critical tasks, such as controlling nuclear power plant. Normally the experiment scenarios can tolerate the delay of Internet traffic. Moreover, normally an instrument has protection mechanisms to prevent it to be damaged by mal-operations. But in order to improve the effectiveness of the online experiments, some extra mechanisms are designed to detect the mal-operations and the lost of connections. These mechanisms can be tuned according to the requirements of a specific experiment. With these mechanisms, the instrument service is more robust and safe to use.

- ***Time out mechanism***. When the user does not give further order, the service cuts the connection.
- ***Attach time stamp when transferring signal***. The time stamp marks the time point of event. The measurement can happen after the event. The time stamp tells the true time of an event.
- ***Trend predication for some critical variables***. The predication can be used to shut down the device when the variables go out of norm values; or it can be used to adjust the control strategies.
- ***Proactively response to exceptions***. These exceptions can be both hardware or software exceptions.

We need to look into the specific experiments to design these mechanisms.

3.3.3 *Reliable Communication*

HTTP is the commonly used transport protocol for Web Services. It is a stateless data-forwarding mechanism which brings two problems to online experiments: no guarantee of packets being delivered to the destination; no guarantee of the order of the arriving packets. If using SOAP message to transfer data to a data processing service, reliable communication is required (Foster, Kesselman, Nick, and Tuecke, 2004). For real-time control, it is often better to deal with some loss, errors or congestion than to try to adjust for them (Salzmann, and Gillet, 2002). In (Salzmann, and Gillet, 2002), the latest data of the sensors are transported instead of re-send the old data. When reliable communication is required in some scenarios, we can use reliable communication protocols such as HTTPR and BEEP instead of HTTP. Another solution is to use asynchronous message services. Message queuing products like Java Messaging Service (JMS) or IBM MQSeries can be used to improve reliability, but at the cost of response time.

4. The performance Issues for Web Services for Online Experiment

The trade-off of the high interoperability of Web Services is the lower performance. Web Services have intrinsic weaknesses on performance due to two main reasons: *more transport layers than middleware*; *the overhead of using SOAP*. Other research explains SOAP efficiency. In this paper, we evaluate SOAP in the context of instrument web services. We benchmark the SOAP efficiency in this context and propose the solutions to improve the performance.

In our system architecture, there are four tiers: the client, the online lab server, the remote web server for the instrument web service, and the remote instrument. The path of transport is as following: client → online lab server → web service server → instrument. On each step, there are some operations, such as buffering, encoding and decoding. Among those, Web Services are only used between the online lab server and the remote web server for the instrument service. Therefore the following analysis concentrates only in this segment.

4.1 The Efficiency of SOAP Message

The SOAP messages are used to transport data. SOAP messages have the XML format. The data are tagged according to its semantic meaning. Here we analyze the efficiency of SOAP payload. We use the oscilloscope as an example. Given different sample rates, the waveform is represented in different amount of data. In Table 3 is the efficiency of the SOAP payload.

Sampling rate (#/cycle)	10	100	500	1000	5000
Pure data (bytes)	191	1,964	9,824	19,632	98,182
Response message (bytes)	3,071	25,085	123,745	247,054	1,241,604
Ratio: data/message	6.22%	7.83%	7.94%	7.95%	7.91%

Table 3: The efficiency of SOAP payload

We can see that the increasing of the SOAP response message is linear to the increasing of the sampling data (which is linear to the sampling rate). In other word, when the sampling rate increases to *k times*, the size of the response message increases to *k* times. The ratio of the data in the SOAP message is pretty low, which is only about 7.9%. See 4.3.1 to know how to improve SOAP efficiency.

4.2 Latency

This benchmark test aims at determining the roundtrip time for a request. The time involves marshalling the SOAP message and binding it to the HTTP protocol at the request side and vice versa on the service side. This test is taken place when the instrument web service and the OES are the same host, i.e. the delay by Internet is not considered. Table 4 shows the latency of request and response SOAP messages. For the request SOAP message, the payload is the same, so the latency for request messages can be taken as the same, within the measurement error. The latency of the response message is increasing when the data amount is increasing. The increasing ratio is close to 1.0.

Sampling rate (#/cycle)	10	100	500	1000	5000
Request latency (ms)	230 ms	221 ms	200 ms	201 ms	201 ms
Response latency (ms)	81 ms	230 ms	471 ms	861 ms	3294 ms
Total latency (ms)	291 ms	451 ms	671 ms	1062 ms	3495 ms

Table 4: The latency of SOAP message

4.3 Optimize the SOAP Efficiency

Currently, people spend a lot of effort to improve the performance of SOAP services. In this section, we present some of the solutions that are workable for our application. We did not finish testing these methods when we wrote this paper. So the effectiveness of the proposed methods is not included in this paper.

4.3.1 *Using Binary Encoding to Improve the Transport Efficiency*

Using binary code violates the initial motivation to use textual XML representations for SOAP message. But it is one of the possible ways to improve the SOAP message efficiency. ASN.1 (Abstract Syntax Notation One) (Sandoz, *et al*, 2003)(Dubuisson, 2004) is a well-established standard that transports the mutually agreed data schema first and then transports the binary data. Fast Web Services is the term that specifies ASN.1 SOAP messages exchanges (encoded using the ASN.1 Packed Encoding Rules, PER) which provide a higher transaction processing rate) than use of the character encoding of XML data.

4.3.2 *Persistent Connection*

This function of HTTP protocol can be turned on so that the connection between client and server can be kept alive. This saves the time to establish connection every time. For HTTP 1.0, the persistent connection works only if there is no proxy between the client and server. For HTTP1.1, the persistent connection can be used with more than one proxy between a client and a server.

4.3.3 *Disable the Nagle Algorithm*

One more issue is the Nagle Algorithm's effect. Nagle Algorithm makes TCP wait for the sender process to fill up an IP packet before sending the data (Litou, 2002). It prevents network congestion. This is the major delay factor when sending a SOAP message (Elfwing, Paulsson, and Lundberg, 2002). It's possible to disable Nagle on both server and client side to get considerable improvement for the response time. But it would generate traffic congestion since repeated small packets will be sent out. However, it's recommended to disable Nagle Algorithms in our case.

4.3.4 *Better Pipelined Connection*

HTTP 1.0 is not well functioning with TCP protocol. For example, the TCP handshake causes delays, and the Nagle algorithm in combination with the TCP delayed ACK (the acknowledge response in TCP) causes unnecessary delays (Litou, 2002). HTTP1.1 attempts to solve these problems. The result shows that HTTP1.1 can reduce the RTT (Round Trip Time) to half of HTTP1.0 implementation (Nielsen, *et al*, 1997).

5. Conclusions

Web services can be used as the middleware to architect the online experiment systems. The trade-off of its high interoperability is its low performance. But SOAP services can be optimized to meet the requirements of this application. Web services will be used in this context, as well as other e-science applications.

References

Andrews, T., F. Curbera, *et al*., (2004), Specification: Business Process Execution Language for Web Services Version 1.1, *http://www-128.ibm.com/developerworks/library/ws-bpel/*

Bagnasco, A., M. Chirico, A. M. Scapolla, (2002) XML Technologies to Design Didactical Distributed Measurement Laboratories, *IEEE IMTC2002*, Anchorage, Alaska, USA.

Dubuisson, O., (2004), ASN.1: A Powerful Schema Notation for XML and Fast Web Services, *http://asn1.elibel.tm.fr/en/xml/ASN1-XML-FastWebServices.htm*

Elfwing, R., U. Paulsson, and L. Lundberg, (2002), Performance of SOAP in Web Service Environment Compared to CORBA, *Proceedings of the Ninth Asia-Pacific Software Engineering Conference (APSE'02)*, 2002, IEEE.

Fattouh, B. and H. H. Saliah, (2004), Model for a Distributed Telelaboratory Interface Generator, *Proceedings of Int. Conf. On Engineering Education and Research*, Czech Republic, June 27-30, 2004.

Foster, I., C. Kesselman, J. M. Nick, S. Tuecke, (2004), The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, *http://www.globus.org/research/papers/ogsa.pdf.*

IEEE Learning Technology Standards Committee, (1999), IEEE 1484 Learning Objects Metadata (IEEE LOM), *http://www.ischool.washington.edu/sasutton/IEEE1484.html.*

Litou, M., (2002), Migrating to Web Services – Latency and Scalability, *Proceedings of Fourth Int. Workshop on Web Site Evolution (WSE'02)*, 2002, IEEE.

Nielsen, H., J, Gettys, A. Baird-Smith, E. Prud'hommeaux, H. Lie, and C. Lilley, (1997), Network Performance Effects of HTTP/1.1, CSS1, and PNG, *http://www.w3.org/Protocols/HTTP/Performance/Pipeline.html*, June 1997.

Paquette, G., (1999), Meta-knowledge Representation for Learning Scenarios Engineering, *Proceedings of AIEd'99*, Le Mans, France, July, 1999.

Saliah-Hassane, H., D. Benslimane, I. De La Teja, B. Fattouh, L. Do, P. Gilbert, M. Saad, L. Villardier, Y. Yan, A General Framework for Web Services and Grid-Based Technologies for Online Laboratories, *iNEER Conference for Engineering Education and Research*, March, 2005, Tainan, Taiwan. (Invited Paper)

Salzmann, C., and D. Gillet, (2002), Real-time Interaction over the Internet, *Proceedings of IFAC2002*.

Sandoz, P., S., Pericas-Geertsen, K. Kawaguchi, M. Hadley, and E. Pelegri-Llopart, (2003), Fast Web Services, *http://java.sun.com/developer/technicalArticles/WebServices/fastWS/.*

UDDU.org, (2004), UDDI homepage, *http://uddi.org/pubs/uddi_v3.htm*

W3C, (2004a), SOAP Specification, *http://www.w3.org/TR/soap12-part1/*

W3C, (2004b), WSDL Specification, *http://www.w3.org/TR/wsdl*